Chapter 1
Introduction to the JUNOScript API

The JUNOScript application programming interface (API) is an Extensible Markup Language (XML) application that Juniper Networks routers use to exchange information with client applications. XML is a metalanguage for defining how to mark the organizational structures and individual items in a data set or document with tags that describe the function of the structures and items. The JUNOScript API defines tags for describing the components and configuration of routers.

Client applications can configure or request information from a router by encoding the request with JUNOScript tags and sending it to the JUNOScript server on the router. (The JUNOScript server is a component of the management daemon [mgd process] running on the router and does not appear as a separate entry in process listings.) The JUNOScript server directs the request to the appropriate software modules within the router, encodes the response in JUNOScript tags or formatted ASCII as requested by the client application, and returns the result to the client application. For example, to request information about the status of a router's interfaces, a client application can send the JUNOScript tag called <get-interface-information>. The JUNOScript server gathers the information and returns it in a tag called <interface-information>.

This manual explains how to use the JUNOScript API to configure Juniper Networks routers or request information about configuration or operation. The main focus is on writing client applications to interact with the JUNOScript server, but you can also use the JUNOScript API to build custom end-user interfaces for configuration and information retrieval and display, such as a Web browser-based interface.

This chapter discusses the following topics:

About XML

XML is a language for defining a set of markers, called tags, that define the function and hierarchical relationships of the parts of a document or data set. The tags look much like Hypertext Markup Language (HTML) tags, but XML is actually a metalanguage used to define tags that best suit the kind of data being marked.

The following sections discuss XML and JUNOScript:

XML and JUNOScript Tags on page 4

Document Type Definition on page 4

For more details about XML, see *A Technical Intr oduction t o XML* at http://www.xml.com/pub/98/10/guide0.html and the additional reference material at the xml.com site. The official XML specification is available at http://www.w3.org/TR/REC-xml.

XML and JUNOScript Tags

JUNOScript tags obey the XML convention that a tag name indicates the kind of information enclosed by the tag. For example, the name of the JUNOScript <interface-state> tag indicates that the tag contains a description of a router interface's current status, whereas the name of the <input-bytes> tag indicates that its contents specify the number of bytes received.

JUNOScript tag names are enclosed in angle brackets, which is an XML convention. The brackets are a required part of the complete tag name, and are not to be confused with the angle brackets used in the JUNOS Internet software manuals to indicate optional parts of command-line interface (CLI) command strings.

When tagging items in an XML-compliant document or data set, you always enclose the item in paired opening and closing tags. XML is stricter in this respect than HTML, which sometimes uses only opening tags. The following examples show paired opening and closing tags:

```
<interface-state>enabled</interface-state>
<input-bytes>25378</input-bytes>
```

If a tag is *empty* —has no contents—you can represent it either as a pair of opening and closing tags with nothing between them or as a single tag with a forward slash after the tag name. For example, the string <snmp-trap-flag/> represents the same empty tag as <snmp-trap-flag></snmp-trap-flag>.

When discussing tags in text, this manual conventionally uses just the opening tag to represent the complete tagged item. For example, it usually refers to "the <input-bytes> tag" rather than "the <input-bytes></input-bytes> tag element."

Document Type Definition

An XML-tagged document or data set is *structured*, because a set of rules specifies the ordering and interrelationships of the items in it. The rules define the contexts in which each tagged item can—and in some cases must—occur. A file called a *document type definition* , or *DTD*, lists every tag that can appear in the document or data set, defines the parent-child relationships between the tags, and specifies other tag characteristics. The same DTD can apply to many XML documents or data sets.

Advantages of Using the JUNOScript API

The JUNOScript API is a programmatic interface. The JUNOScript DTDs fully document all options for every command and all elements in a configuration statement. JUNOScript tag names clearly indicate the function of an element in a command or configuration statement.

The combination of meaningful tag names and the structural rules in a DTD makes it easy to understand the content and structure of an XML-tagged data set or document. JUNOScript tags make it straightforward for client applications that request information from a router to parse the output and find specific information.

The following example illustrates how the JUNOScript API makes it easier to parse router output and extract the needed information. It compares formatted ASCII and XML-tagged versions of output from a router. The formatted ASCII follows:

```
Physical interface: fxp0, Enabled, Physical link is Up
   Interface index: 4, SNMP ifIndex: 3
```

This is the JUNOScript-tagged version:

```
<interface>
   <name>fxp0</name>
   <admin-status>enabled</admin-status>
   <operational-status>up</operational-status>
   <index>4</index>
   <snmp-index>3</snmp-index>
</interface>
```

When a client application needs to extract a specific piece of data from formatted ASCII output, it must rely on the datum's location, expressed either absolutely or with respect to adjacent strings. Suppose that the client application wants to extract the interface index. It can use a utility such as expect to locate specific strings, but one difficulty is that the number of digits in the interface index is not necessarily predictable. The client application cannot simply read a certain number of characters after the Interface index: label, but must instead extract everything between the label and the subsequent string:

```
, SNMP ifIndex
```

A problem arises if the format or ordering of output changes in a later version of the software, for example, if a Logical index field is added following the interface index number:

```
Physical interface: fxp0, Enabled, Physical link is Up
   Interface index: 4, Logical index: 12, SNMP ifIndex: 3
```

A search for the interface index number that relies on the SNMP ifIndex string now returns an incorrect result. The client application must be updated manually to search for the following string instead:

```
, Logical index
```

In contrast, the structured nature of the JUNOScript-tagged output enables a client application to retrieve the interface index by extracting everything within the opening <index> tag and closing </index> tag. The application does not have to rely on an element's position in the output string, so the JUNOScript server can emit the child tags in any order within the <interface> tags. Adding a new <logical-index> tag in a future release does not affect an application's ability to locate the <index> tag and extract its contents.

Tagged output is also easier to transform into different display formats. For instance, you might want to display different amounts of detail about a given router component at different times. When a router returns formatted ASCII output, you have to design and write special routines and data structures in your display program to extract and store the information needed for a given detail level. In contrast, the inherent structure of JUNOScript output is an ideal basis for a display program's own structures. It is also easy to use the same extraction routine for several levels of detail, simply ignoring the tags you do not need when creating a less detailed display.

Overview of a JUNOScript Session

Communication between the JUNOScript server and a client application is session-based: the two parties explicitly establish a connection before exchanging data and close the connection when they are finished. The streams of JUNOScript tags emitted by the JUNOScript server and a client application each constitute a *well-formed* XML document, because the tag streams obey the structural rules defined in the JUNOScript DTDs for the kind of information they encode. Client applications must produce a well-formed XML document by emitting tags in the required order and only in the legal contexts.

The following list outlines the basic structure of a JUNOScript session. For more specific information, see "Start, Control, and End a JUNOScript Session" on page 14.

1. The client application establishes a connection to the JUNOScript server and opens the JUNOScript session.

2. The JUNOScript server and client application exchange initialization tags, used to determine if they are using compatible versions of the JUNOS Internet software and the JUNOScript API.

3. The client application sends one or more requests to the JUNOScript server and parses its responses.

4. The client application closes the JUNOScript session and the connection to the JUNOScript server.